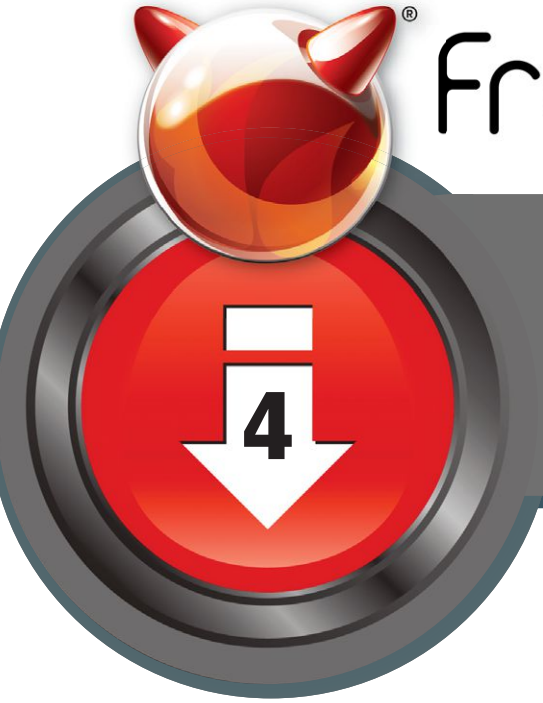
FreeBSD® JOURNAL
Jan/Feb 2019

Getting Started with FreeBSD

FreeBSD® JOURNAL
Jan/Feb 2019



A GUIDE TO Getting Started with FreeBSD

Getting started is easy and there is a variety of choices as to where you can install and run FreeBSD. **By Roller Angel**

8 Getting Started with JAVA Development in FreeBSD/GhostBSD

A short introduction to setting up your computer running FreeBSD 12.0 or other desktop operating systems that derive from it—such as GhostBSD or Trident desktop. **By Ashik Salahudeen**

10 FreeBSD for Developers

FreeBSD is a complete operating system and everyone from system administrators to software developers to accountants can use it for daily jobs. **By Mariusz Zaborski**

14 Writing Manual Pages

The source code of a BSD operating system includes an unrivaled man page collection that accompanies the installed programs. This article covers the basics for writing new or improving existing man pages for most BSD systems. **By Aaron St. John**

18 UPSTREAMING A Document Bugfix

The FreeBSD documentation set consists of a large number of documents, man pages, and websites that are great ways for newcomers to get involved in the community and contribute something back to the operating system. **By Benedict Reuschling**

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Copy Editor • Annaliese Jakimides

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

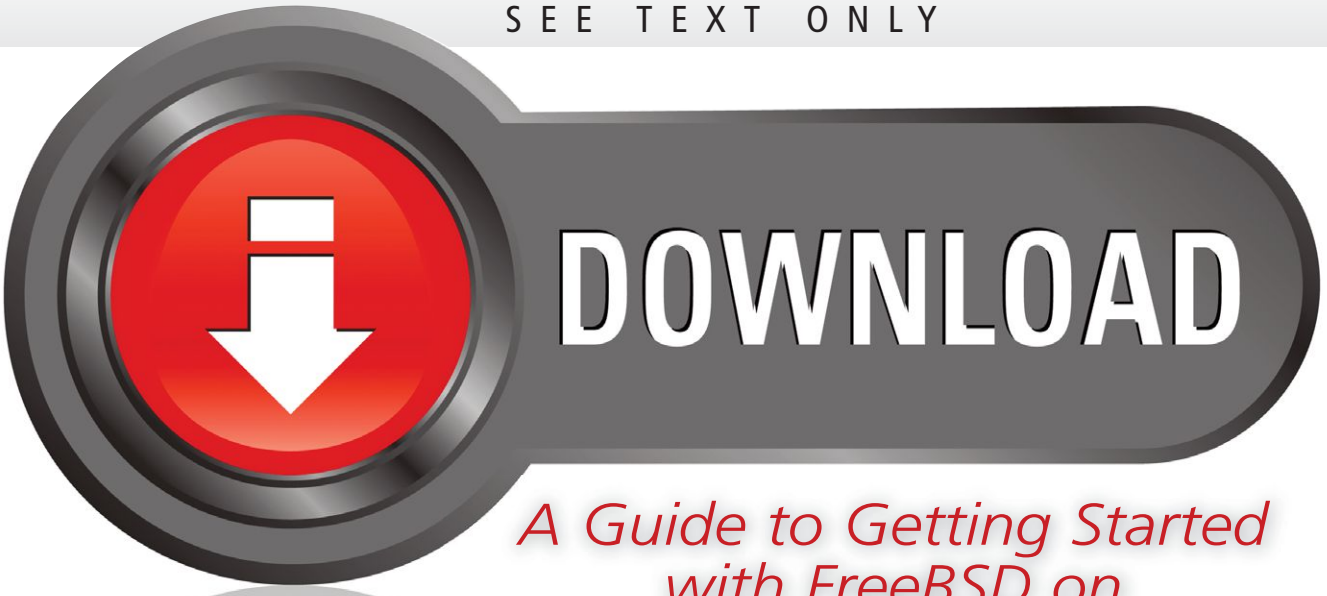
FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
2222 14th Street, Boulder, CO 80302
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

Welcome to the January/February 2019 Issue of FreeBSD® JOURNAL

As we announced in the last issue, ***FreeBSD Journal*** is now a ***Free*** subscription. Readers can access and share current and past issues for free. That's not all. We're also moving away from the Apple and Google mobile apps and debuting an enhanced, browser-based edition that will provide a much better experience for readers using any device—mobile or otherwise—for reading *FreeBSD Journal*. Beginning with the March/April 2019 issue, a new options menu will allow you to easily share articles with friends and listen to the articles with the new narration feature. Plus, jumping from issue to issue will be easier than ever with the new carousel format. If you currently view *FreeBSD-Journal* on one of the mobile apps, please ignore any renewal reminders you may receive and check out our FAQ (https://www.freebsdfoundation.org/journal/freebsd-journal-faqs/) for more information on how to sign up for the free, browser-based edition.

This issue's focus is Getting Started with FreeBSD. What better way to share that message than by making *FreeBSD Journal* freely available! We look forward to bringing *FreeBSD Journal*'s timely and informative articles to a larger audience and hope you'll share your favorites with colleagues and friends.

Anne Dickison
FreeBSD Foundation



A Guide to Getting Started with FreeBSD on Virtual and Real Hardware

By **Roller Angel**

Getting started with FreeBSD is easy and there is a variety of choices as to where you can install and run FreeBSD. We will stick to two common-use cases, installation in a virtual machine and installation on real hardware— in this case an amd64 desktop computer.

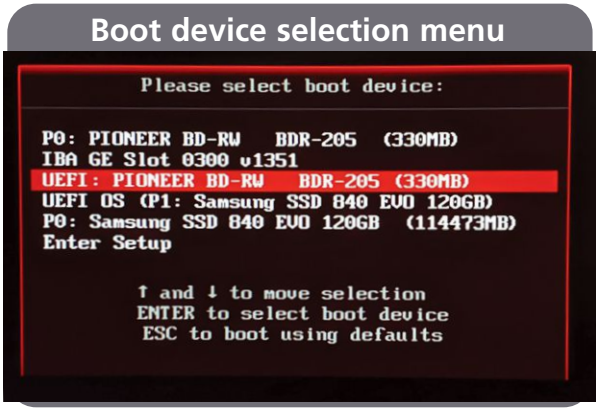
To begin, we can simply visit FreeBSD.org on a working computer and look for the big "Download FreeBSD" button near the awesome beastie logo. Once on that page, you will need to select the proper image from the list of "Installer Images." We'll choose "amd64." Next, we'll pick the image called FreeBSD-12.0-RELEASE-amd64-bootonly.iso from the list because it's the latest RELEASE version of FreeBSD and is quite small. I highly recommend you also take a look at the CHECKSUM.SHA512-FreeBSD-12.0-RELEASE-amd64 file and do your own SHA512 checksum verification after you have downloaded the file to see if they match. If so, you know that your download worked properly.

Once the installer image has been downloaded, it is ready to be used to boot up the virtual machine. On real hardware, there are a few extra steps. Turn the .iso into a real CD by burning it and selecting a boot device on startup. We'll assume your real hardware has a CD Drive and that you burned the .iso file to a CD and put it into the drive. When you first boot the computer, a BIOS screen controls which device is used to start the machine. To change this device to the installer CD, you may need to consult the manual for the computer. On most computers, you can try booting and then immediately pressing F8 repeatedly until a "Boot Device Selection" menu appears. Select the proper device from the list to start the Installer. We selected the UEFI version of the CD device, in this case, a Pioneer BDR-205 disc reader/writer.

In VirtualBox, click the "New" button to start the

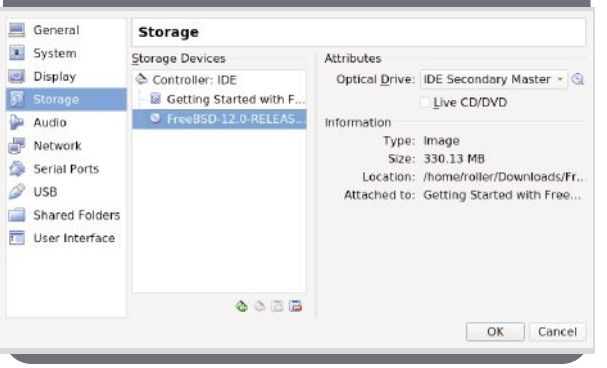
Real hardware BIOS screen

Boot device selection menu

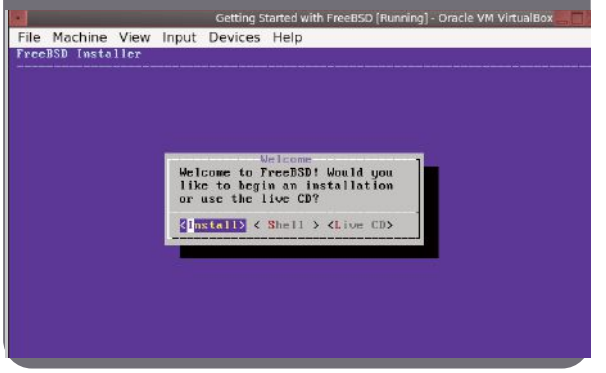


wizard that will walk you through creating a virtual machine. The important options to keep in mind are the type of OS—we'll choose FreeBSD 64 bit, and the amount of RAM to dedicate to the virtual machine. Use some amount greater than 512MB and less than any number that puts your virtualbox wizard "slider" into the red. Keep your "slider" in the green, and you should be fine giving the machine a decent amount of memory to work with. Accept the defaults for the other settings. Once the wizard completes, you can start up the virtual machine. Another wizard will appear asking you to point VirtualBox to the installer image for the new virtual machine. If you'd like to avoid this second wizard, simply edit the virtual machine settings before first boot, clicking on the "Storage" link, then the "Empty Disc" section, and then you can mount the virtual image on that screen. All of these steps in VirtualBox equate to assembling the hardware and inserting the installer CD. Pressing "Start" in VirtualBox is like pressing the "Power" button to boot up a real machine.

Mount a CD in VirtualBox



FreeBSD Installer



On first boot of the install CD you are welcomed to FreeBSD and asked if you would like to begin an installation. Select "Install" by pressing "Enter/Return" on the keyboard.

At this point, you will have noticed VirtualBox trying to explain it is capturing your mouse and keyboard to be used inside the virtual machine. Go ahead and dismiss these messages. To use the mouse and keyboard outside of the virtual machine again, you need to press the host key. VirtualBox displays the current host key in the bottom right of the window. This install says "Right Ctrl," which means you need to press the right control button on the keyboard first to get back control over the mouse and keyboard from the virtual machine.

Continue through the installer with the following in mind: for this install, we'll continue with the default US keyboard layout, name the computer "getting-started-with-freebsd," and select the default settings on the "Distribution Select" screen by just pressing "ok." For the "Network Configuration" section, we went with the "em0" interface on the virtual machine and the "re0" interface on the real hardware. Yes, configure IPv4. Yes, use DHCP. Yes, use IPv6. Yes, try SLAAC. The resolver configuration should be populated with at least one DNS value. Use the "Tab" key on the keyboard to move to select "ok" to continue. For the "Mirror Configuration," we used "Main Site." Next, we need to decide how to partition the disc. On the "Partitioning" screen, there are options for Auto (ZFS) and Auto (UFS).

• ZFS Version

Select Auto (ZFS), choose stripe as the ZFS pool type and select the disc to use for the pool with the space-bar, name the pool, and proceed with installation. Finally, agree to install FreeBSD by selecting "Yes."

• UFS Version

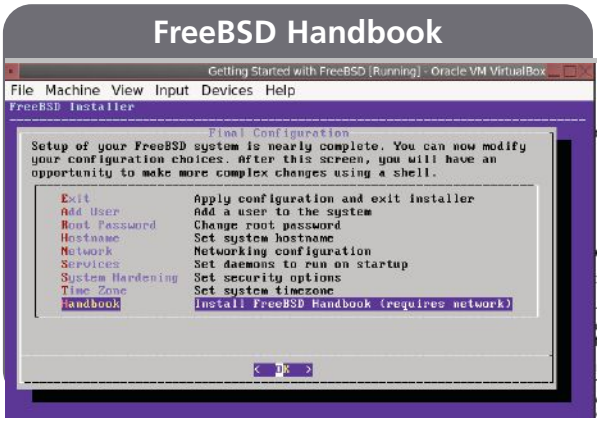
Select Auto (UFS), use the entire disc, select GPT for the partition scheme, and select "Finish" when shown the disk layout. Finally, agree to erase the computer and install FreeBSD by choosing "Commit."

• User Accounts

Set a password for the root account on the system. Don't fret when there are no *** appearing while typing in your root password. Rest assured, the keystrokes are being recorded; they're just not being shown on the screen.

For the "Time Zone" section we went with "America – North and South, United States of America," "Mountain (most areas)," "Skip" on both "Time & Date" screens.

For the "System Configuration" section, we unselected "sshd," selected "powerd" and chose "ok." On the next screen, we selected "9 secure_console" to put the root password prompt on any attempt to enter Single User Mode at the system console.



Say "yes" to the prompt about adding a new user. Use an all lowercase username. Enter the users "Full name." Add the user account to the groups wheel operator video by responding to the question "Login group is .. Invite .. into other groups? []:" with **`wheel operator video`**, then press "Enter." Users in the wheel group can run "privileged" commands. The operator group allows users to do things like shut down and restart the computer without needing to invoke the special privileges from the wheel group. It's a good idea to add users to the video group that is planning on using FreeBSD as a desktop. We chose "tcsh" for our shell and used defaults for the other fields while finally providing an account password. If all looks ok, proceed without creating any more user accounts.

For the "Final Configuration" section, select "Handbook." We went with the default of "en" and chose "ok." Once back on the configuration screen, select "Exit" to finish and "yes" to proceed with doing manual configuration. The only manual configuration we need to do is to shut down the machine so we can take out the CD. On the real machine, you will need power to take out the CD, so go ahead and remove the CD now. On the virtual machine, the power needs to be off to not get an error when removing the CD, so we'll issue the shutdown command below to turn off the computer.

Type either "init 0" or "shutdown -p now" to shut down.

To take a CD out of a virtual machine in VirtualBox, click on the name of the virtual machine, click on "Settings," click on "Storage," and click on the tiny CD icon. Notice another tiny CD icon with a tiny down arrow below it next to the "Optical Drive" section. Use this second CD icon to "Remove Disk from Virtual Drive." Now "Start" the machine and continue.

Package Installation / Text Configuration

Boot up the computer and log in as the root user.

Type the following commands on the virtual machine:

```
pkg install -y xorg sudo lumina sakura virtualbox-ose-additions firefox
sysrc vboxguest_enable=YES
sysrc vboxservice_enable=YES
```

Type the following command on the real machine:

```
pkg install -y xorg sudo lumina sakura firefox
```

Type the following commands on either machine:

`visudo` (this command launches the vi text editor).

Type the following to edit the one line of text and get out of the editor:

```
/wheel
```

Type enter or hit return and then type

`j0xxZZ` (once you hit the first search term, j goes down, 0 goes to the beginning of the line, xx deletes the comment, ZZ saves and quits vi)

```
sysrc dbus_enable=YES
dbus-uuidgen > /etc/machine-id
service dbus start
```

Change to regular user by holding down "control" and pressing "d" on the keyboard which will log out the current user. Log in again, this time as the regular user account that was created. Next type the following commands.

```
vi .xinitrc
```

Again the vi editor, and to begin typing you need to be in "insert mode" and must press "I" to enter insert mode and begin typing in text. Type in the following one-line configuration:

```
exec start-lumina-desktop
```

Exit vi by pressing "Esc" on the keyboard, then type ZZ or :wq followed by "Enter/Return."At this point the virtual machine desktop can be started with the following command

```
startx
```

To get the real machine working, you will need to choose a display driver that works with the real hardware. We went with a basic non-3D accelerated graphics setup using the scfb display driver. To set it

up, type the following:

```
vi /usr/local/etc/X11/xorg.conf.d/
driver-scfb.conf
```

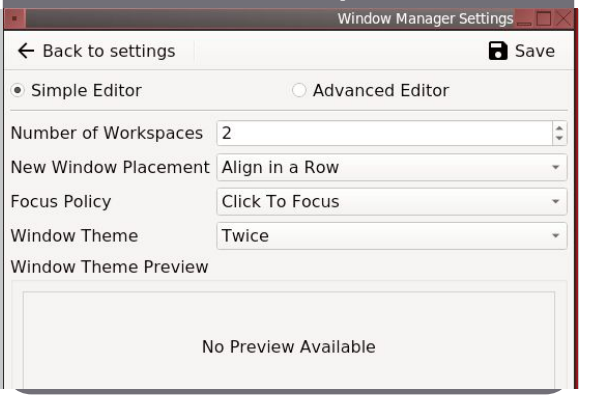
Add the following lines of configuration to the file using "Esc" ZZ to exit when finished.

```
Section "Device"
  Identifier "Card0"
  Driver "scfb"
EndSection
```

Now the real hardware machine's desktop can be started with the following command:

```
startx
```

Lumina desktop theme



Desktop Configuration

Right click on the desktop > Preferences > All Desktop Settings

Under Appearance click on "Theme," then select "Icons" from the sidebar, then select "Material Design (light)," and click on "Apply." Close the "Theme Settings" window.

Go back to the "Desktop Settings" window and select "Window Manager." In the "Window Theme" drop-down menu, select "Twice." Click on "Save." Close the "Window Manager Settings" window.

Go back to the "Desktop Settings" window and select "Applications." Set the "E-mail Client" to Firefox and "Virtual Terminal" to Sakura. Close all the settings windows.

Right click on the desktop and select "Preferences" then "Wallpaper," click on the "+," and choose "solid color." We chose red for our color. Click "ok," then "Save."

FreeBSD Handbook

Here is how to open a local copy of the handbook we installed in the Final Configuration:

Look for Firefox in the applications menu in Lumina.

Right click on the desktop > Applications > Network > Firefox Web Browser

Open Firefox and navigate to the following URL: file:///usr/local/share/doc/freebsd/handbook/index.html

This is a local copy of the FreeBSD Handbook, and it will always be available as it doesn't require an active internet connection.

That should get you started using FreeBSD. The next recommended step is to choose a firewall and configure it. There is a great FreeBSD Journal article in the May/June 2014 issue called "IPFW An Overview," which will help guide you through the process. Also, pick up a copy of the book *BSD Hacks* to get more familiar with the "tcsh" shell as well as to acquire several new skills. ●

***ROLLER ANGEL** is an avid BSD user who enjoys all the amazing things that can be done with BSD technology. He has taught programming workshops based on FreeBSD and is working on building an online training platform for teaching BSD and related technologies. See BSD.pw for more information.*

Getting Started with Java Development in FreeBSD/GhostBSD

By *Ashik Salahudeen*

This is a short introduction to setting up a Java development environment on your computer running FreeBSD 12.0 or other desktop operating systems that derive from it — such as GhostBSD or Trident desktop.

What Is Java Anyway?

Java is a general-purpose, object-oriented programming language developed at Sun Microsystems. It was first released in 1996. Its primary motto was "Write once, run everywhere," which makes it possible for the developers to write Java code, compile it into byte code, and be able to execute it on any operating system that had a Java runtime. So, someone can write code on Windows, but execute it on FreeBSD or GNU/Linux servers. Over time, Java has become extremely popular, and is a very reliable choice for writing server-side applications as well as Android applications.

Java on the FreeBSD Platform

Oracle Corporation bought Sun and is now owner of the official Java runtime and development kit implementations. They provide implementations for Windows, Mac OSX, and Linux operating systems. The official reference implementation is open-sourced under GPLv2 (with a linking exception), and hence it is possible to have OpenJDK implementations for FreeBSD.

The current long-term support version of Java is Java 11, released in September 2018. There are no builds for FreeBSD yet. The previous stable release was Java 8 released in March 2014 and builds for this version exist. The rest of this article is based on Java 8 openjdk.

Getting Started: Install the Java Development Kit

To write applications in Java, you will need a Java Development Kit that provides a compiler, a runtime, and a standard library.

Thankfully, this is available as a pre-built package, and for development purposes, the binary package is sufficient. Install this using the following command (as root):

```
pkg install openjdk8
```

This should not take a long time, and you may verify that the installation is successful by invoking that Java compiler from a terminal.

```
$ javac -version
javac 1.8.0_181
```

Editing Java Code

You can edit Java code using a plain text editor, but because of how the language operates, it gets cumbersome very soon. Almost all Java programmers use an IDE to edit their code. There are several editors available and some of them are commercial. The following is a list of recommended IDEs:

Intellij IDEA Community Edition

IntelliJ is an excellent IDE, available as a binary package and is released under the Apache20 license. Install it using:

```
pkg install intellij
```

The makers of this IDE, Jetbrains, also make a commercial (paid) version called IntelliJ Ultimate, which has more features. For all practical purposes, the community edition should suffice.

Netbeans

Netbeans is another well-built IDE that has been around for a long time. It is also released under the Apache20 license and is available as a pre-built package. Install it using:

```
pkg install netbeans
```

Eclipse

Eclipse is another popular IDE, available under the Eclipse Public License. It is also available as a pre-built package. Install it using:

```
pkg install eclipse
```

Some Generic Configuration to Make Things Easier

In general, setting some environment variables/configurations will make Java development easier. The following is a brief list:

Java Home

This variable specifies the location of your JDK/JRE. It is useful to set this in your environment variable via the .profile file. Add the following line:

```
JAVA_HOME=/usr/local/openjdk8
```

Maven Home

Maven is a popular project management tool for Java projects. It is a pure Java application and is available via the binary packages. The project gets updated often though, so if you want to get the latest Maven binary, it is still best to download it from the project site directly. If you choose to do this, set up the location to Maven's installation directory using the environment variable M2_HOME. This path is the location of "bin" directory in Maven's installation.

```
M2_HOME=/path/to/apache-maven-3.x.0/
```

Intellij – Fix Font Rendering

If the fonts in the IDE look fuzzy, you can fix the problem by passing in the following arguments to the launcher script. Intellij makes it easy via the "Help > Edit Custom VM Options" menu. Paste the following in the text file that opens up. Close the IDE and launch it again.

```
-Dsun.java2d.renderer=sun.java2d.
 marlin.MarlinRenderingEngine
-Dawt.useSystemAAFontSettings=on
-Dswing.aatext=true
-Dsun.java2d.xrender=true
```

Netbeans – Fix Font Rendering

Netbeans' font rendering can be fixed by passing the same arguments, but it does not provide an easy way to do this. As root (or sudo) open "/usr/local/netbeans-8.2/etc/netbeans.conf" in a text editor. If you are on a newer version of Netbeans, just change "8.2" in the above to that version. Find the line that sets the variable "netbeans_default_options" and append the following to the end of the option string. It is one long line.

```
-J-Dswing.aatext=true -J-Dawt.use
SystemAAFontSettings=on -J-Dsun.
java2d.renderer=sun.java2d.marlin.
MarlinRenderingEngine -J-Dsun.java2d.
xrender=true
```

Summary

These things should get you set up for developing Java applications, but this article does not go into detail about writing a Java program or customizing the IDEs or various options you can pass to the JVM when you run it. You can contact me on aashiks@gmail.com if you have queries. ●

ASHIK SALAHUDEEN *has been working with computers for about 18 years, mostly with Unix-like operating systems and open source tech. He runs the engineering team at Accordium and works with other FOSS communities during his spare time—primarily the Indic language computing group, Swathanthra Malayalam Computing.*

by Mariusz Zaborski

FreeBSD for Developers

When someone mentions FreeBSD, you probably think about your servers and NAS. Without a doubt, FreeBSD is a great operating system for those purposes. However, there are some myths out there that FreeBSD isn't good for developers and that if you want to do software development on or of FreeBSD, you should get a Mac. The truth is that FreeBSD is a complete operating system and everyone from system administrators to software developers to accountants can use it for daily jobs. In this article let's look into FreeBSD as software developers.

Languages

The most important question for software developers is whether their language is supported by FreeBSD. Well, all the popular languages are supported by FreeBSD. There is no limit, really. Do you want to do your shiny new web in Node.js, PHP, or Python? FreeBSD supports all of them. If you are looking more for low-level programming like Go or Rust, FreeBSD has it. Are you classier than that? Then you can find a new version of clang and gcc in the ports. You can even work with Java through OpenJDK or C# through Mono.

What is great is that in many cases the native FreeBSD package manager pkg(1) permits you to install the required compiler. In the case of PHP, Ruby, or Python, many *packages* are also provided by *pkg*. If you want to install pygame for Python, you can simply type:

```
# pkg install py{27,36}-game
```

This allows you to manage all the packages you have installed in your operating system. It's easy and clean. If, for some reason, you don't like this approach, you can still use pip as always. It is the same with WordPress. If you run the command below you'll be ready to go:

```
# pkg install wordpress
```

The full list of packages with language compilers and interpreters can be found by executing:

```
# pkg search lang/
```

It's also great that all packages are up-to-date. The repology project (https://repology.org/repositories/statistics/newest) is doing an analysis of a huge number of package repositories and other sources comparing package versions across them. The conclusion is that FreeBSD has one of the newest ports. Its packages are far more updated, for example, than the newest version of Ubuntu. If you are looking for an up-to-date language infrastructure, FreeBSD is your choice.

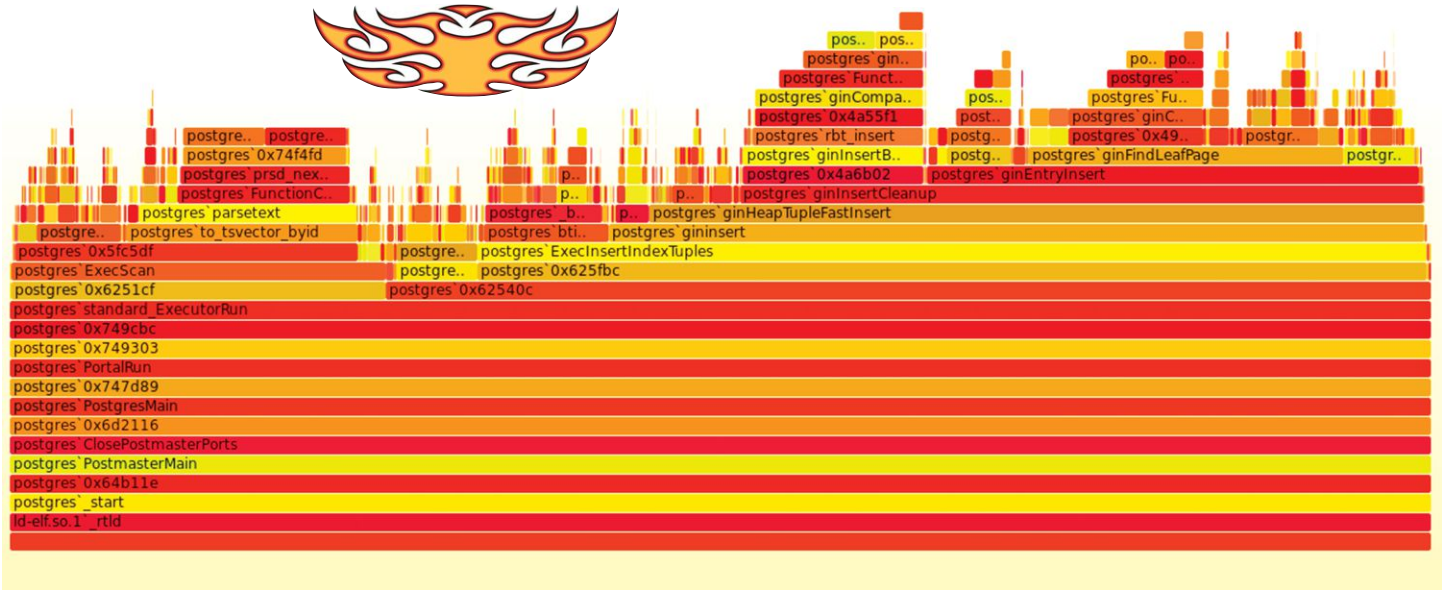
Developer's Environment

Another important question for software developers is whether their favorite IDE will work on FreeBSD. When we talk about a developer environment, you probably associate Unix with vim and emacs. That's true, and you can use them with FreeBSD, although you will also find more modern IDEs for your work like:

- Eclipse
- Sublime Text
- Pycharm (freeware and commercial version)
- IntelliJ (ultimate and community)

If your IDE does not support FreeBSD directly, you can always try to run them using Linuxulator, a Linux compatibility layer in FreeBSD. You also don't need to worry about your code version control system. You will find all popular, open-source VCS like git, mercurial, or svn. You also may use commercial ones like perforce.

There is no better way to maintain your envi-



Example of a flame graph generated from PostgreSQL, generated using output from DTrace one liner:

```
# dtrace -x ustackframes=100 -n 'profile-5000 { ustack(); }` -p PID -o output
```

ronment and test your software than jails. If you want to have a few versions of your application for different customers or multiple databases and so on, then jails are what you want to use. You can also combine them with ZFS and easily upstream your changes to production. These days, managing jails is easier than ever by using *iocage* or *ezjail*.

Sometimes jails are not enough and we have to work with different operating systems. FreeBSD also has this covered. If you are looking for lightweight virtualization of any modern operating system like Windows, Linux, OpenBSD, NetBSD or FreeBSD *bhyve* is the way to go.

Contain Directory Madness

Where do you install your software in Linux: */usr/bin, /bin, /usr/local/bin*? Or maybe *bin* is a simple symlink to */usr/bin* and everything is a total mess? The administrative binaries always land in the *sbin*; the normal binaries land in *bin*. Also, the hierarchy of directories is clean and understandable:

- */bin* and */sbin* are the FreeBSD-specific binaries shipped with the operating system that are required for minimal work with the system,
- */usr/bin* and */usr/sbin* are the FreeBSD-specific binaries shipped with operating systems, and
- */usr/local/bin* and */usr/local/sbin* are directories where all your third-party software will land.

Additionally, you can read about the whole filesystem hierarchy on the manpage **here(7).** Again, FreeBSD wins with its simple and clean solution.

Look into Your Program

One of the best tools for developers, which is lacking in other operating systems, is DTrace. We don't have space to describe all functionalities of DTrace but once I started using DTrace, I don't know how I coped without it. DTrace is a dynamic tracing framework and was originally developed by Sun Microsystems. Using a few lines of the D script, we are able to see a current stack of the program, analyze the performance, trace the input of the functions, and much, much more.

The real power of DTrace is in the few scripts written by Brendan Gregg (https://github.com/brendangregg/FlameGraph) that allow you to generate flame graphs. Flame graphs are a visualization of traced software allowing the most frequent code paths to be identified. You can use these to trace many things, starting with your software, FreeBSD kernel, and ending with third parties. Everything that you need is a binary with some symbols. To use it, you do not need to stop your program or recompile it. We recently used DTrace on the PostgreSQL database to trace why the insert was taking so much time. Thanks to DTrace, we easily found the problem with table settings and were able to fix it. This is a big advantage over other operating systems for software developers, and DTrace should be in your toolbox.

Other tools that may be useful for you when developing low-level applications, and which are FreeBSD-specific, are:

- ***procstat*** – a powerful tool that allows you to get a lot of process information
- ***ktrace/kdump*** – an alternative for *strace(1)*, which allows you to print a list of called syscalls
- ***pmccontrol*** – control hardware performance monitoring counters
- ***pmcstat*** – performance measurement with performance monitoring hardware

In the case of low-level debugging by default, you will not find a gdb in the base system any

more, but you can use a great alternative lldb from the LLVM project, which we encourage you to give a try.

Some Great APIs That the World Needs to Catch Up With

FreeBSD is a pioneer in many technologies. It was the first operating system to catch up with ZFS—which took the Linux world ages. FreeBSD first saw the potential in containers a long time before docker—developing *chroot* and *jails*. FreeBSD is POSIX compatible, but it also introduces or integrates many great APIs that are not available in many popular operating systems.

All operating systems implement poll and select, which have problems with scalability. We also see epool, which, instead of solving problems, introduces some. FreeBSD took a different direction and introduced *kqueue(1)*. *kqueue* provides efficient input and output event pipelines between the kernel and userland. *kqueue* is much more efficient, especially when polling for events on a large number of file descriptors.

In talking about descriptors, we should also mention process descriptors—a new way to add a handle (descriptor) to the process. Process identifiers (PID) used by many operating systems are not reliable. Between checking the status of process and sending signals, many things can occur in the operating system, and, in theory, the PID may be used by another process. Process descriptors solve those problems by giving you a reliable handle to add to the process. If the process is terminated, you still will have a handle to inform you about it.

And there is the FreeBSD sandboxing technique called Capsicum. Capsicum is a lightweight OS capability and sandbox framework. Capability-based security means that processes can only perform actions that have no global impact. Processes cannot open files by their absolute path or cannot open network connections. The idea is to protect your application with the process privilege separation in mind.

To secure your applications, you may also consider using CloudABI (https://github.com/NuxiNL/cloudabi). CloudABI takes a POSIX function and adds capability-based security and removes everything that's incompatible with that. This forces software developers to use very specific sets of functions in their applications but increases the security of the application.

FreeBSD is working on many interesting technologies that in the future may be standard in other operating systems. If you want to be up-to-date you should look into this OS.

Documentation

FreeBSD is known to have great documentation. In the case of software development, this is no different. How many times have you googled for an ASCII table? In FreeBSD, by default, you have a manpage for it: **ascii(7)**. The same goes for architecture/language-specific things like:

- **arch(7)** – architecture-specific details like size of the pointer, numbers, or pages.
- **operator(7)** – C and C++ operator precedence and order of evaluation.
- **zstyle(9)** – the best C style you will find – FreeBSD has used it for decades.
- **hier(7)** – to understand Unix filesystems layout.

The Sky's the Limit

FreeBSD has a lot of interesting features like jails, bhyve, ZFS, and DTrace that make life easier for software developers. You also will find very useful documentation in the system. This is not to mention the large amount of third party software that is just waiting to be used. One of the great ways of learning new languages is to create some games in it—and, of course, you can do this on FreeBSD! So, don't wait a moment longer—configure your new developer box on FreeBSD! ●

MARIUSZ ZABORSKI is a QA&Dev manager at Fudo Security. He has been the proud owner of the FreeBSD commit bit since 2015. Mariusz's main areas of interest are OS security and low-level programming. At Fudo Security, Mariusz leads a team that is developing the most advanced solution for monitoring, recording, and controlling traffic in IT infrastructure. In 2018, Mariusz organized the Polish BSD User Group. In his free time, he enjoys blogging at https://oshogbo.vexillium.org.

Groundbreaking TrueNAS® M Series

DISRUPTING STORAGE AT WARP SPEED

LOWEST TCO SHARED STORAGE

Intel® Xeon® Scalable Family Processors

RESILIENT

- Self-healing Data
- Continuous Operation
- Easy Replication

WARP FACTOR IX

- Faster than SSD-based Hybrid Storage Arrays
- Flash-Turbocharged Data Access

EXPANDABLE

- Scales to 10PB using HGST Drives
- 10Gb/s-100Gb/s per NAS
- Non-disruptive Upgrades

COMPATIBLE

- Citrix, Veeam, & VMware Certified
- Unifies File, Block, & S3 Data
- Supports Leading Cloud Providers

Visit **iXsystems.com/TrueNAS** or call **(855) GREP-4-iX** today!

HGST a Western Digital brand

iXsystems

Writing Manual Pages

By Aaron St. John

Manual (man) pages served as some of the first forms of documentation available for Unix operating systems. They still are used as quick-reference guides on how to use certain computer programs that are installed on the system. The source code of a BSD operating system includes an unrivaled man page collection that accompanies the installed programs. Generally, man pages include an EXAMPLES section, a very useful tool, that includes common usage of the command or function. Creating or updating an existing man page is simple yet helpful for the entire open-source community. This article covers the basics for writing new or improving existing man pages for most BSD systems.

Manual Page Sections

The BSD manual pages are divided into sections. The extension of the file represents the section index for the man page. Manuals are sorted into different sections based on type. The nine manual page sections are shown in Table 1. Usually there are only one or two man pages for a particular command, syscall, or file.

Table 1

Section	Description
1	General commands executed by users
2	System calls; functions that wrap operations performed by the kernel
3	Library functions
4	Kernel interfaces
5	File formats
6	Games
7	Miscellaneous information
8	System manager
9	Kernel developer

Writing a Manual Page

Markup

Over the years several different renderers such as groff(7) and mandoc(1) have been used with man pages. FreeBSD has used roff(7), troff(1), and man(7) markup languages in the past. However, new man pages and most existing ones use mdoc(7), a semantic markup language that uses macros. A line beginning with '.' is known as a "macro line." Following the '.' are two or three letters referred to as the macro name. Begin a macro name with a capital letter and lowercase for the remaining letters. Lines that do not begin with '.' are known as "text lines," providing free-form text to be printed. Common practice when writing multiple sentences is to begin a new line for each sentence which provides clarity to the reader. Comment lines begin with .\". Here is an example of properly formatted macros.

```
.Sh NAME
.Nm examplecommand
```

Layout

Manual pages can be written many different ways. However, man pages usually contain specific sections to ensure consistency. First, the man page must contain a prologue with the macros **.Dd**, **.Dt**, and **.Os** in that order.

```
.Dd $Mdocdate$
.Dt PROGNAME section
.Os
```

The macro **.Dd** is the date macro. When making a change to an existing man page, the date must be updated. The date can be manually updated by typing the date after the macro in the format of *month day, year*. **.Dt** is the document title macro. This macro is followed by the name of the command or function and its section. Lastly, the **.Os** macro specifies the operating system in

use. The system can be manually specified. However, using **.Os** without any arguments is recommended. A list of frequently used macros is shown in Table 2.

Table 2

Macro	Description
.Dd	Document date: *month, day, year*
.Dt	Document title: *TITLE section*
.Os	Operating system version: [*system [version]*]
.Sh	Section header (one line)
.Nm	Command or function name
.Nd	Command or function description
.Op	Optional syntax arguments
.Ar	Command arguments
.Bl, **.El**	Begin list and end list
.It	List item
.Pp	Start a text paragraph
.An	Author name

The standardized sections that must be included in man pages are:

• NAME

Contains the name of the function or command and a concise one-line description of what it does.

• SYNOPSIS

If it is a command, write any options that can be used with the command. If it is a program function, write a list of parameters the function can use, and which header file contains the definition. Here is a correctly formatted portion of the SYNOPSIS section for iocage(8):

```
.Sh SYNOPSIS
.Nm
.Op Fl help | Ar SUBCOMMAND Fl help
.Nm
.Op Fl v | version
.Pp
.Nm
.Cm activate
.Ar ZPOOL |
```

• DESCRIPTION

The description contains a concise but complete write-up of the command or function.

• EXAMPLES

List of use cases describing what each case does. A robust EXAMPLES section contains at least one trivial, everyday, and inspirational use case.

Manual pages are not limited to these sections alone. In fact, most man pages have many more sections. Some sections that are commonly used are explained in Table 3.

Table 3

Common Sections	Description
ENVIRONMENT	Environment settings that affect operation
EXIT STATUS	Error codes returned on exit
COMPATIBILITY	Compatibility with other implementations
SEE ALSO	Cross-reference to related manual pages
STANDARDS	Compatibility with standards like POSIX
HISTORY	History of implementation
BUGS	Known bugs
AUTHORS	People who created the command or wrote the manual page.

Example

A trivial example might look like this:

```
.Dd January 22, 2019
.Dt examplecommand 1
.Os
.Sh NAME
.Nm examplecommand
.Nd This is an example command for an example man page.
.Sh SYNOPSIS
.Nm examplecommand
.Op Fl help
.Op -l
.Op -o Ar file
.Sh DESCRIPTION
This is formal text describing this command.
This command does this and that.
It can be used for this and that.
.Pp
These options are available:
.Pp
.Bl -tag -width ".Cm activate"
.It Fl help
List the help screen for the command.
.It Fl l
Does this and that.
.It Fl o Ar file
Opens a file.
.El
.Sh EXAMPLES
Open a file called helloworld.txt.
.Pp
.Dl $ examplecommand -o helloworld.txt
```

The output would be generated as:

```
examplecommand(1) FreeBSD General Commands Manual  examplecommand(1)


NAME
  examplecommand — This is an example command for an example man page.

SYNOPSIS
  examplecommand [--help] [-l] [-o file]

DESCRIPTION
  This is formal text describing this command. This command does this and that. It can be used for
  this and that.

  These options are available:

  --help    List the help screen for the command.

  -l        Does this and that.

  -o file   Opens a file.


EXAMPLES
   Open a file called helloworld.txt.

  $ examplecommand -o helloworld.txt

  FreeBSD 12.0-RELEASE January 22, 2019 FreeBSD 12.0-RELEASE
  (END)
```

For a more everyday example, here is a portion from the ls(1) man page:

```
.Dd December 1, 2015
.Dt LS 1
.Sh NAME
.Nm ls
.Nd list directory contents
.Sh SYNOPSIS
.Nm
.Op Fl libxo
.Op Fl ABCFGHILPRSTUWZabcdfghiklmnopqrstuwxy1,
.Op Fl D Ar format
.Op Ar
.Sh DESCRIPTION
For each operand that names a
.Ar file
of a type other than
directory,
.Nm
displays its name as well as any requested,
associated information.
For each operand that names a
.Ar file
```

Continues next page

Continued

of type directory,
.Nm
displays the names of files contained
within that directory, as well as any requested, associated
Information.

When rendered with **man ls** it displays as:

```
LS(1)       FreeBSD General Commands Manual       LS(1)


NAME
 ls — list directory contents

SYNOPSIS
 ls [--libxo] [-ABCFGHILPRSTUWZabcdfghiklmnopqrstuwxy1,] [-D format]
    [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls displays its name as well as
 any requested, associated information. For each operand that names a file of type directory, ls
 displays the names of files contained within that directory, as well as any requested, associated
 information.
```

Conclusion

At first, writing man pages from scratch might seem daunting. However, after a little research, the mdoc(7) markup language is easy to use. Manual pages are an essential part of a program and the EXAMPLES section is a tremendous help. •

Aaron St. John *works at iXsystems. He is a recent graduate with a BS in Mathematics and a passion for all things technology and video games.*

Sources

Dzonsons, Kristaps. "man(7)." *FreeBSD*, April 5, 2018, www.freebsd.org/cgi/man.cgi?query=man&sektion=7.

"Manual Pages." *FreeBSD*, www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/manpages.html.

"man page." *Wikipedia*, Wikimedia Foundation, January 5, 2019, en.wikipedia.org/wiki/Man_page.

Wirzenius, Lars. "Writing Manual Pages." *Writing Manual Pages*, November 10, 2010, liw.fi/manpages/.

Dzonsons, Kristaps. "mdoc(7)." *FreeBSD*, July 28, 2018, www.freebsd.org/cgi/man.cgi?query=mdoc&sektion=7&manpath=freebsd-release-ports.

Toth, Peter, and Brandon Schneider. "iocage(8)." *FreeBSD*, April 20, 2017, www.freebsd.org/cgi/man.cgi?query=iocage&sektion=8.

Dzonsons, Kristaps, and Ingo Schwarze. "roff(7)." *FreeBSD*, April 10, 2018, www.freebsd.org/cgi/man.cgi?query=roff&apropos=0&sektion=0.

Upstreaming

A DOCUMENTATION BUGFIX

By **Benedict Reuschling**

The FreeBSD documentation set consists of a large number of documents, man pages, and websites. These are great ways for newcomers to get involved in the community and contribute something back to the operating system. Changes are submitted via Phabricator when there is already a patch available that doc committers can review. Other ways to report bugs are the mailing list (freebsd-doc@freebsd.org) and of course Bugzilla.

One such bug was recently reported in https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=231994. It did not provide much of a description but had the right keywords to pique my interest. The subject line read "sudoers repeated word," which meant that it was for the widely-used security/sudo port. Repeated words are a common mistake when people are writing documentation in a hurry and/or don't proofread (or don't have a second set of eyes to do that). The first thing I did was confirm that the reported error was in fact still present in the sudoers file in the most current version. Sometimes people report bugs that are already fixed in the HEAD revision of the port, simply because they did not update to the latest version or someone corrected the error in the meantime. In this case, there was, indeed, a repeated "and and" in the sudoers manual page.

Since FreeBSD does not maintain the sudo software itself (except maybe some local patches), the right way to approach this is not to just fix it locally in our own project, but to report the issue upstream. That way, not only does the sudo project know about the issue, it can look for a solution. Once the issue has been fixed and a corrected version released, all other projects (including FreeBSD) receive these fixes when they install the newer version. There is a much wider impact than just fixing the problem locally.

The sudo project, which can be found at https://www.sudo.ws/, also has a bugzilla instance where people can report bugs. Creating an account there was simple enough and once I confirmed my account via the usual sign-up confirmation email, I could create an issue. While doing that, I thought about the fact that repeated words seldom appear alone, so I looked for other instances in the man page. After a brief search, I found other occurrences. Then I realized that by manually reading the man page, I might miss an instance.

A couple years ago, Warren Block wrote a perl-based tool called igor, the friendly lab assistant, (http://www.wonkity.com/~wblock/igor/) that can be run over man pages and DocBook XML pages to check for various issues. One such check is for repeated words, and, fortunately, the tool is flexible enough to be used outside of FreeBSD. As a result, I ran igor on the sudoers man page and it reported issues other than just repeated words. Getting more and more suspicious that this might also be the case for other man pages in the sudo project, I ran igor on those as well and got a number of hits for other problems.

The sudo project maintains various man page formats with slightly different syntax, so I had to make individual changes to those. The nice thing about man page changes is that they can be viewed immediately by running the man(1) program and passing the changed man page to it as a parameter. No need for lengthy compiles, so you'll get quick feedback for the changes you made. Once I had fixed all issues reported by igor (including the originally reported one that got us here), I created a patch using diff -ruN. The patch was attached to the issue https://bugzilla.sudo.ws/show_bug.cgi?id=854 in sudo's bugzilla with a description and link to the igor tool.

It did not take long until Todd Miller took the patch and integrated it (https://www.sudo.ws/repos/sudo/rev/4ddcb625f3b7). He was also intrigued by igor and wrote in the issue: "Thanks, I've added an 'igor' target to the doc Makefile and will fix the things it finds that seem like problems." This means that in the future, each time there are documentation changes in sudo, igor will run as part of the man page and may report issues. That way, quality issues can be fixed before they hit the src repository and are not distributed in the next release.

Meanwhile, a new sudo version 1.8.26 was released that contained the documentation fixes, and they were even mentioned in the release notes (https://www.sudo.ws/stable.html#1.8.26). The FreeBSD port was updated shortly after (https://svnweb.freebsd.org/ports?view=revision&revision=484929) and so the fixes are available as part of FreeBSD too. That way, what started out with a small error reported to a downstream project consuming sudo turned into a larger patch for multiple doc issues reported and fixed upstream. This happens all the time in open-source projects and not just for documentation. The same thing is done for src and ports and is a great demonstration of collaboration and exchange of fixes, tools, and ideas that benefit everyone. The nice thing is that it is easy to start with documentation work, especially man pages. Small fixes are as worthwhile as big improvements and, as you saw, may lead to bigger things than initially meet the eye.

People interested in starting out with documentation work should take a look at the FreeBSD Documentation Project primer (https://www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/). The quickstart section explains everything that is needed in terms of tools and getting the sources. Once you have that, you can start bug hunting in the docs. Installing the textproc/igor port is simple using FreeBSD's pkg tool. Once you find a bug, make sure that it has not been reported or fixed as yet. If it has not, be sure to report it to the project and people that are actually maintaining it. Many projects have a bug tracking system or a mailing list where issues can be reported. Make sure to include as much information as possible and a clear description along with any patches you might have already created. This increases the chance that your patch will be looked at and taken care of. If you have any questions about the FreeBSD documentation process, you can ask them on the freebsd-doc mailing list or find us on IRC in the #bsddocs channel on Efnet. •

***BENEDICT REUSCHLING** joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he actively began mentoring other people to become FreeBSD committers. He is a proctor for the BSD Certification Group and joined the FreeBSD Foundation in 2015, where he is currently serving as vice president. Benedict has a Master of Science degree in Computer Science and is teaching a UNIX for software developers class at the University of Applied Sciences, Darmstadt, Germany.*

conference REPORT

by Benedict Reuschling

MeetBSD 2018 and FreeBSD Developer Summit

Conference Recap

MeetBSD 2018 was held at Intel's Santa Clara Campus in California. I was looking forward to the event for a number of reasons: first, I was part of the program committee that selected the talks for the main conference; second was the venue at the Intel Campus; and third, it provided a good opportunity to see and talk to some people in the BSD community that I normally only see once a year at BSDCan.

On the day I arrived, I met Michael Dexter, Rod Grimes, and Michael Lucas for dinner. Dan Langille joined us a bit later. The Mexican food was excellent, and it was great to talk about various topics in a smaller circle. The next morning, Deb Goodkin and Anne Dickison drove me and the rest of the Foundation folks (Ed Maste, Scott Lamons, and Li-Wen Hsu) to the Intel Campus.

Arriving at the venue, I was greeted by JT, our BSDNow.tv producer and he handed me my badge and conference bag before I could take a look around. I also met Denise Ebery, who had put a lot of effort into organizing the event and making sure everything was running smoothly. The first day before the main conference, there was a one-day FreeBSD Devsummit sponsored by the FreeBSD Foundation. Intel provided us with a large auditorium (which reminded me more of a cinema), a podium for giving the talks, and an area next to the entrance for food and drinks. You could basically spend the whole day in there without missing anything except for bathroom breaks. Before the talks started, there was plenty of opportunity to meet and greet developers, and it didn't take long for us to engage in familiar conversations about BSD, travel, and tech.

The devsummit talks started with *LWPMFS: LightWeight Persistent Memory Filesystem* by Ravi Pokala. The topic was interesting and completely new to me. After a short coffee break, Ed Maste continued with *Evaluating GIT* for FreeBSD, covering the arguments for and against it. I think he managed not to incite too much heated discussion, while still engaging people in considering the various points. Discussions continued well into the lunch break. During the break, I got a chance to talk with Devin Teske and saw the updates she'd done to her dwatch tool since her well-received presentation at BSDCan in July 2018. Mark Johnston gave a talk about *NUMA* after the lunch break, detailing the progress the project has made in scalability and increases in performance. After another coffee break, we had the famous "Have, Need, Want" session to hear what people would like to see (or could offer) as features in the next major FreeBSD release. While this may sound boring, John Baldwin and George Neville-Neil, who hosted the session, are a unique combination that make these sessions both entertaining and engaging.

The next day, *MeetBSD* started in the same venue, but with a lot more attendees. After an introduction round led by JT and Michael Dexter, Kris Moore showed us *How to Use TrueOS to Bootstrap Your FreeBSD-based Project*, detailing some of the things he and his team at iXsystems (that sponsored the event and made the Intel venue possible) had done. I always find it interesting to see the different ways people can transform the operating system for various purposes. Next, Ben Widawsky presented *Intel & FreeBSD: Better Together*, a very engaging talk about his way into the community and what kinds of opportunities there are in the partnership with Intel.

The nice thing about this MeetBSD was that it wasn't just talks the whole day. The single-track conference style made it possible for everyone to stay in the same room and we were able to enjoy discussions, lightning talks, panels, and other fun ways of engaging with each other. For me, this made the day seem much less rushed without having to wait for the next break to talk with someone.

The social event that evening was held in Intel's cafeteria just a few doors down from our venue in

the same building. Another great opportunity to mingle and talk to people over pizza and drinks. It was a lot of fun and it felt good to be within the BSD crowd yet again.

The next day began with the *Why BSD?* keynote by Michael W Lucas as only he could give it: funny, entertaining, and thoughtful. Kirk McKusick made a surprise appearance as the next speaker and even allowed us to pick which talk we would like to hear by a show of hands. *The Early History of BSD* won and Kirk delivered a talk that does not seem to get boring, even though you might have heard it before. I guess that's the mark of a good speaker.

After my third variety of lunch box (each day I tried a different one and could never decide which was best), G. Clifford Williams delivered his talk about *A Curmudgeon's Language Selection Criteria*. While there are a lot of languages you might learn over the years as a programmer, there is a tendency to use your most familiar and try to apply it to every problem you encounter. I see that a lot in my students, and Mr. Williams elaborated some excellent points to be aware of and how to not fall into this trap.

Before the virtualization panel discussion, Mariusz Zaborski delivered his talk on *Capsicum*. He's done a lot of work converting FreeBSD utilities one by one to this sandboxing framework. The talk he gave provided a good introduction to Capsicum and gave pointers for people who want to help out in that area.

Nick Principe told us cautionary tales about doing *Performance Measurements*, which is part of his job at iXsystems. I thought the talk was well done, with good slides and a lot of lessons about proper performance benchmarking. We closed with a group picture and thanked all the organizers and sponsors for making MeetBSD such a success. The FreeBSD 25th anniversary party was held in the Intel cafeteria, and it was definitely a celebration. During the evening, some people took the microphone and told us what FreeBSD and its community meant to them. It was a very heartwarming celebration, and I had a chance to talk to some new people that night. I returned home very satisfied, having participated in an amazingly successful conference. It was an excellent way to conclude my BSD conference travels for this year. •

BENEDICT REUSCHLING joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he actively began mentoring other people to become FreeBSD committers. He is a proctor for the BSD Certification Group and joined the FreeBSD Foundation in 2015, where he is currently serving as vice president. Benedict has a Master of Science degree in Computer Science and is teaching a UNIX for software developers class at the University of Applied Sciences, Darmstadt, Germany.

WeGetletters

by Michael W Lucas

Hey Letters Column Flunky,
What's with all the firewalls? Will we ever get rid of any of them? And will you really answer any questions we send in?
Thanks,
Troublemaker

Dear Troublemaker,

Yes, I'll answer any question you ask, so long as it survives review by *FreeBSD Journal*'s esteemed editorial board. Mind you, they won't let me use words like "pusillanimous" and "mewling," so my answer might not be as useful as you might hope. They'd almost certainly reject "lily-livered," especially if I used it in reference to them, so I won't.

In fairness, I have my own rejections.

I wholeheartedly reject your question. The word "firewall" means *nothing*.

If you thawed my primordial Unix mentor from his cryogenic capsule (and handled the humdrum minutia like fixing all the cancer and starting his heart and sealing all the cells burst from ice crystals because homebrew cryogenics really translates to serious post-mortem freezer burn—especially after that three-day Great Blackout of 2003 probably drained his UPS), he wouldn't recognize anything we call a "firewall." I delved into antediluvian mailing lists to try to find the first firewall on the Internet, exercising a smidge of effort you certainly won't appreciate nearly enough, and found myself wholly blocked by this ambiguity.

A firewall started off as a type of non-flammable physical wall. Put a firewall between two buildings and you could set one to the torch without burning down the other, which must have been really convenient for the Huns when they wanted the fun of sacking Rome and setting the temples ablaze before pillaging the treasury next door. At least, that's what my predecessors told me. My 1933 Oxford English Dictionary doesn't include the word "firewall" and Oxford University knew all about Rome, so I'm guessing the early Internet engineers just made up that etymology to see if we'd believe them.

Today we've settled on a couple different approaches to firewalls: the packet filter and the proxy.

A packet filter regulates which connections can pass. You can configure a host's packet filter to protect that host or drop a packet filter in front of a whole network to control IP-level access to the network. Packet filters must be integrated with the kernel, unless you treat performance with the contempt normally reserved for politicians. FreeBSD ships with three: IPFW, IP Filter, and PF.

A proxy terminates all TCP/IP connections to the outside world, inspects the traffic at a higher level of the application stack, and originates a new request. FreeBSD includes dozens of these critters in the packages collection. A search of the ports index gives 981 proxies, and while I'm sure a bunch of those aren't actually proxies, I can't be bothered to audit the whole list, so let's go with the tediously well-known standards like Squid, SOCKS, and relayd. In a previous millennium, I made a decent living installing and supporting the FireWall Tool Kit, the primordial proxy. In my off hours, I amused myself by creating droll retronyms for FWTK.

Youngsters who use words like *devops* and *serverless* think that packet filter firewalls are the whole deal. Then their blockchain dotcom crashes. They scramble to secure insufficiently gainful employment and suffer seizures when confronted with proxies. Many globe-trampling firms disallow all unproxied connections to the Internet in the name of regulatory compliance, data control, or some nebulous hallucination of "security." Opening a direct TCP/IP connection out of one of these firms resembles splenic auto-extraction via the sinuses.

How do all of these firewally *things* get in FreeBSD?

Because someone maintains them.

Why do they maintain them?

Because they *need* them.

Nobody spends what few precious minutes our overhurried lives leave unallocated getting bludgeoned by code they don't need. I supported mod_auth_xradius for a few years because I desperately needed it to glue Apache to the company's authentication system. It was either maintain a port or run everything on the compa-

ny platform, which I won't name but is alliterative with Abominable Dysentery, so I learned to send patches and deal with Bugzilla and all that, which, while occasionally frustrating, beat blue bile out of forcibly extracting useful information from Event Obscurer.

While I'm here, let me recommend FWTK. It's still online at fwtk.org. Release 2.1 came out on February 27, 1998, although a second 2.1 escaped on March 2, 1998, because we hadn't yet invented proper release versioning. FWTK is why I applauded the arrival of Squid and IPFW, which are why I celebrated IP Filter, which is why I threw a festival for the appearance of PF and relayd.

That last release is now old enough to drink and gamble in Vegas.

In related news, I'll be in Vegas on March 2, 2019. Perhaps I should throw FWTK a coming-out party.

Dear Letters Column Flunky,
I meant the packet filters, you silly goose. And, if you'll answer any question: What's the difference between a poorly-dressed programmer on a unicycle and a well-dressed programmer on a bicycle?

Troublemaker

Dear Troublemaker,

Again: it's because people need them.

Any code in FreeBSD, kernel or userland, needs care and feeding. Programmers get these daft ideas like "support new hardware" and "nobody uses twoax any more," so they constantly change code internals and APIs in the name of progress. Change the network stack to support 40GB Ethernet cards and someone has to assess the packet filter code to see if it still works.

If nobody tweaks that code, eventually it no longer works and someone—traditionally, a Dane—axes it from the tree.

IPFW is the primordial FreeBSD firewall. It's a favorite among many senior developers who learned it in the late nineties and don't see why anyone would want anything simpler. I've used it to simulate a transoceanic link in a local office, because web developers should suffer the same fate as their hapless users.

IP Filter is for those condemned souls who must use a single packet filter on multiple flavors of Unix. I don't know what they did to be sentenced to multiplatform torment, but it must have been appalling even by my exquisitely high standards.

PF is by far the most popular general-purpose packet filter. It was ported from OpenBSD and then forked to handle FreeBSD's kernel locking, so don't trouble yourself to ask the maintainers when a new import from OpenBSD will happen. It won't. My repeated but wholly unscientific surveys show that roughly 80% of FreeBSD users who run packet filters use PF.

PS: Attire.

Michael W Lucas (https://mwl.io) is the author of too many books, including the brand new third edition of *Absolute FreeBSD*, *PAM Mastery*, and *Butterfly Stomp Waltz*. George Neville-Neil bribed him to write this column and Lucas is still awaiting payment. Send your questions to letters@freebsdjournal.com. Letters will be answered in the order in which they befuddle, betray, or bewilder the columnist, and might be edited for his own beguilement. ●

svn UPDATE

by Steven Kreuzer

I hope your new year is off to a wonderful start and that you had some time over the holidays to upgrade your machines to FreeBSD 12. While I think it is safe to say that the time and effort put into FreeBSD in 2018 was nothing short of remarkable, I have a strong suspicion that 2019 is going to be an even more exciting year for development. We are not even a full month into this year, and I am already seeing some very interesting changes being committed to HEAD. I am eagerly looking forward to what the future has in store and I hope you are feeling the same.

vmm(4): Mask Spectre feature bits on AMD hosts — https://svnweb.freebsd.org/changeset/base/343166

For parity with Intel hosts, which already mask out the CPUID feature bits that indicate the presence of the SPEC_CTRL MSR, do the same on AMD. Eventually, we may want to have a better support story for guests, but for now, limit the damage of incorrectly indicating an MSR we do not yet support.

nvdimm: Add a driver for the NVDIMM root device — https://svnweb.freebsd.org/changeset/base/343143

The NVDIMM root device is parent to the individual ACPI NVDIMM devices. Add a driver for the NVDIMM root device that can own enumeration of NVDIMM devices as well as NVDIMM SPA ranges that the system has.

vmm(4): Take steps towards multicore bhyve AMD support — https://svnweb.freebsd.org/changeset/base/343075

Vmm's CPUID emulation presented Intel topology information to the guest, but disabled AMD topology information and in some cases passed through garbage. For example, CPUID leaves 0x8000_001[de] were passed through to the guest, but guest CPUs can migrate between host threads, so the information presented was not consistent. This could easily be observed with 'cpucontrol -i 0xfoo /dev/cpuctl0'.

Slightly improve this situation by enabling the AMD topology feature flag and presenting at least the CPUID fields used by FreeBSD itself to probe topology on more modern AMD64 hardware (Family 15h+). Older stuff is probably less interesting. I have not been able to empirically confirm it is sufficient, but it should not regress anything either.

Fix bhyve's NVMe Completion Queue entry values — https://svnweb.freebsd.org/changeset/base/342762

The function which processes Admin commands was not returning the Command Specific value in Completion Queue Entry, Dword 0 (CDW0). This affects commands such as Set Features, Number of Queues which returns the number of queues supported by the device in CDW0. In this case, the host will only create 1 queue pair (Number of Queues is zero based). This also masked a bug in the queue counting logic.

Create new EINTEGRITY error with message "Integrity check failed" — https://svnweb.freebsd.org/changeset/base/343111

An integrity check such as a check-hash or a cross-correlation failed. The integrity error falls between EINVAL that identifies errors in parameters to a system call and EIO that identifies errors with the underlying storage media. EINTEGRITY is typically raised by intermediate kernel layers such as a filesystem or an in-kernel GEOM subsystem when they detect inconsistencies. Uses include allowing the mount(8) command to return a different exit value to automate the running of fsck(8) during a system boot.

These changes make no use of the new error, they just add it. Later commits will be made for the use of the new error number and it will be added to additional manual pages as appropriate.

Add support for marking interrupt handlers as suspended — https://svnweb.freebsd.org/changeset/base/342170

The goal of this change is to fix a problem with PCI shared interrupts during suspend and resume.

I have observed a couple of variations of the following scenario. Devices A and B are on the same PCI bus and share the same interrupt. Device A's driver is suspended first, and the device is powered down. Device B generates an interrupt. Interrupt handlers of both drivers are called. Device A's interrupt handler accesses registers of the powered-down device and gets back bogus values (I assume all 0xff). That data is interpreted as interrupt status bits, etc. So, the interrupt handler gets confused and may produce some noise or enter an infinite loop, etc.

This change affects only PCI devices. The pci(4) bus driver marks a child's interrupt handler as suspended after the child's suspend method is called and before the device is powered down. This is done only for traditional PCI interrupts, because only they can be shared.

Optimize RISC-V copyin(9)/copyout(9) routines — https://svnweb.freebsd.org/changeset/base/343275

The existing copyin(9) and copyout(9) routines on RISC-V perform only a simple byte-by-byte copy. Improve their performance by performing word-sized copies where possible.

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retro-computing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.

new faces of FreeBSD

BY DRU LAVIGNE

This column aims to shine a spotlight on contributors who recently received their commit bit and to introduce them to the FreeBSD community. In this installment, the spotlight is on Thomas Munro, who received his src bit in October.

Tell us a bit about yourself, your background, and your interests.

• I am a hacker and committer on the PostgreSQL project, and I'm employed by EnterpriseDB. I've been doing open-source work from my base in Wellington, New Zealand, for about four years. Before that I did a decade or two of proprietary Java, Lisp, C and C++ work on many flavors of Unix, mostly in Europe, mostly in finance/trading technology. I'm interested in finding ways to make PostgreSQL and FreeBSD work together better in terms of performance, correctness, security, and observability.

How did you first learn about FreeBSD and what about FreeBSD interested you?

• I ran into FreeBSD years ago in various contexts, but I first installed it on my own hardware just three years ago because I wanted ZFS for a home storage box. I found myself spending more and more time logged into that machine to do all kinds of work, because it was more fun there. Before long I was using FreeBSD for several projects that needed web servers and database servers. I was also curious about kqueue and dtrace. FreeBSD is easy to use, well documented, actively developed, easy to contribute to, and has a really good book for curious newbie kernel hackers (*The Design & Implementation of the FreeBSD Operating System*). It has a critical mass and a vibrant community, but at the same time there is enough low-hanging fruit for new people to make meaningful contributions. Then there is the rich history of BSD and its gigantic contributions to the technology and culture of our industry. Finally, I can't deny that installing it for the first time felt like a small act of rebellion.

How did you end up becoming a committer?

• Once I had switched to FreeBSD as my main development environment for everyday work on PostgreSQL, I wanted to modify it to provide missing things. I managed to get four patch-sets committed: PROC_PDEATHSIG_CTL, setproctitle_fast(), shm_open()/shm_unlink() support for truss, and expose_authtok for pam_exec.so. Humble beginnings and in quite different areas, but they scratched an itch relating to work I was doing, helped me learn the ropes and put me in contact with a couple of committers. I have a shopping list of further patches in development, and I was talking to one of my future mentors about some of them on Freenode IRC. He put my name forward as a potential committer. I accepted.

How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming a FreeBSD committer?

• It's early days, and my involvement will be sporadic due to other commitments, but I was happy to commit a patch submitted via bugzilla to fix a 30-year-old bug in pom(6). I'm also excited to make it to the BSD devroom at FOSDEM, which will be a first for me.

As for becoming a committer, I can only say what worked for me: find weak points in FreeBSD for your workload or area of interest. Hack. Submit patches. Respond to feedback and pay attention to the Project's way of doing things. Make your plans and interests known and talk to people. •

DRU LAVIGNE is a FreeBSD doc committer and the author of *BSD Hacks* and *The Best of FreeBSD Basics*.

2019 Events Calendar

BY DRU LAVIGNE • **The following BSD events will take place during March and April 2019**

SCALE • March 7–10 • Pasadena, CA

http://www.socallinuxexpo.org/scale/17x • SCaLE 17X, the 17th annual Southern California Linux Expo, will take place on March 7–10, 2019, at the Pasadena Convention Center. SCaLE 17X expects to host 150 exhibitors this year, along with nearly 130 sessions, tutorials, and special events including a *Getting Started with FreeBSD* workshop.

FOSSASIA 2019 • March 14–17 • Singapore, Singapore

http://2019.fossasia.org/ • The FOSSASIA Summit, Asia's leading Open Technology conference for developers, start-ups, and IT professionals, will take place March 14–17, 2019, at the Lifelong Learning Institute Singapore.

AsiaBSDCon • March 21–24 • Tokyo, Japan

https://2019.asiabsdcon.org/ • This annual conference is for anyone developing, deploying and using systems based on FreeBSD, NetBSD, OpenBSD, DragonFlyBSD, Darwin and MacOS X. AsiaBSDCon is a technical conference and aims to collect the best technical papers and presentations available to ensure that the latest developments in our open-source community are shared with the widest possible audience.

MENOG 19 • March 31–April 4 • Beirut, Lebanon

https://www.menog.org/meetings.menog-19/ • Members of the Foundation team will be participating in MENOG 19, taking place March 31–April 4, 2019. The event will consist of 2 parallel workshops held over 3 days, followed by 2 plenary days on topics including IPv6 deployment, Network and DNS operations, Network and routing security, Content delivery, Internet peering and mobile data exchange, and more.

Gothenburg FreeBSD Community Day • April 7 • Gothenburg, Sweden

https://wiki.freebsd.org/DevSummit/201904 • Join fellow FreeBSD community members at a FreeBSD Community Day in Gothenburg, Sweden. Taking place April 7, 2019, the event will be held in conjunction with the FOSS-north 2019 conference. FOSS-north is a 2-day open-source conference with a technical focus, 2 tracks and speakers from all over the world.

Aberdeen FreeBSD Hackathon • April 17–19 • Aberdeen, Scotland

https://wiki.freebsd.org/Hackathon/201904 • Join us for a FreeBSD Hackathon at the University of Aberdeen in Scotland. This event is placed at the halfway point between AsiaBSDCon and BSDCan to give ardent hackers something to fill their time. More information can be found on the wiki. The registration deadline is March 31, 2019.